IN-62-CR
217912
13P.

Speeding Up Parallel Processing

Peter J. Denning

23 May 1988

RIACS Technical Report TR-88.15

NASA Cooperative Agreement Number NCC 2-387

(NASA-CR-184646) SPEEDING UP PARALLEL PROCESSING (Research Inst. for Advanced Computer Science) 13 p CSCL 09B

N89-25628

Unclas
G3/62 0217912

RIACS

Research Institute for Advanced Computer Science

Speeding Up Parallel Processing

Peter J. Denning

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report TR-88.15
23 May 1988

In 1967 Amdahl expressed doubts about the ultimate utility of multiprocessors. His formulation, now called Amdahl's law, became part of the computing folklore and has inspired much skepticism about the ability of the current generation of massively parallel processors to efficiently deliver all their computing power to programs. The widely publicized recent results of a group at Sandia National Laboratory -- which showed speedup on a 1024 node hypercube of over 500 for three fixed size problems and over 1000 for three scalable problems -- have convincingly challenged this bit of folklore and have given new impetus to parallel scientific computing.

This is a preprint of the column *The Science of Computing* for *American Scientist 76*, No. 4 (July-August 1988).

Work reported herein was supported in part by Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA).

Speeding Up Parallel Processing

Peter J. Denning

Research Institute for Advanced Computer Science

23 May 1988

In 1986, Alan Karp of the IBM Scientific Center in Palo Alto issued a challenge -- a reward of $100 to anyone who could design a practical program that runs more than 200 times faster on a parallel processor than on a single processor (*1*). This bound is equivalent to an assumption, arising from a theory called Amdahl's law, that practical programs contain at least half a percent of speed-limiting sequential operations. Karp obviously did not believe that the limit of 200 was unbreakable -- after all, he only put $100 of his own money on the line -- but he did want to stimulate algorithm developers to search for practical ways to break this barrier.

Gordon Bell, then the head of the Computer and Information Science Directorate at NSF, added to the incentives in 1987, when he offered two annual awards of $1000 of his own money to promote parallel processing research leading to a massive speedup of practical programs. The first award was given on

March 1 of this year to John Gustavson, Gary Montry, and Robert Benner of Sandia National Laboratory, who had demonstrated a near-perfect speedup for three scientific problems on an NCUBE/ten, a hypercube with 1024 processors (*2*).

Within days of the award, major national newspapers ran stories about the "breakthrough in supercomputing" that had been achieved at Sandia, and even the normally sober magazine *Science* got caught up in the frenzy (*3*). These articles suggested that a basic law of computing (Amdahl's) had been disproved and that the result had taken the computing research community by surprise. What is Amdahl's law, and how startling to computer scientists was the Sandia result?

In 1967, Gene Amdahl published a paper in which he expressed doubts that multiprocessors could significantly speed up practical programs (*4*). Theoretically, one could expect a speedup no greater than the number of processors: if a problem requires N operations, and if each of P processors could independently perform an equal portion of those operations, the whole program could be completed in N/P operation times. The difficulty with this line of reasoning, Amdahl observed, is that the instructions of a program are not independent; for example, the sum of two numbers cannot be computed until both numbers have been previously computed. The best possible running time is determined by the longest sequential path in the program. If that path contains S operations, the best possible speedup is

$$\frac{\text{Time with 1 processor}}{\text{Time with } P \text{ processors}} = \frac{N}{S+(N-S)/P},$$

which can never exceed N/S no matter how large P is. Thus the sequential part of the program is an inherent bottleneck blocking parallel speedup.

Known as Amdahl's law, this formulation has been part of the computing folklore for many years now. In 1984, when the first commercial multiprocessors containing hundreds or thousands of processors were announced, the latent skepticism about the practical utility of massively parallel processors came to the fore. Practical programs contain many sequential steps in addition to those in their kernel algorithms; these include the steps to obtain parameters, set up the computation, load the program into the machine, gather the results, and create displays. In a program designed explicitly for a parallel machine, the sequential steps also include operations for sending and receiving messages between processors; the total of these operations constitutes the communication time of the program. Although many kernel algorithms can be speeded up proportionately with more processors, it is not obvious whether many practical programs can be speeded up by much.

Despite the widespread acceptance of Amdahl's law outside the computing research community, the community itself did not consider the doubts expressed in 1967 as binding on computer performance two decades later. Far from being surprised by the development at Sandia, many researchers were hot on the trail of demonstrating an almost linear speedup of important practical algorithms.

Substantial sums of money from NSF, DOD, DOE, and other agencies are invested in research in "scalable algorithms," and this research has now begun to bear fruit.

What did the Sandia group accomplish? What has been learned from their work? I will comment on these questions, and I also recommend (highly) that you read their paper (*5*).

The Sandia group considered two approaches to using a parallel processor. The goal of the first is to solve a fixed-size instance of a problem faster; as the number of processors grows, the piece of the problem assigned to any one of them shrinks, but the ratio of communication time to computation time rises and limits the speedup. The figure of merit, which is called fixed-size speedup, is constrained by Amdahl's law. The goal of the second approach is to solve the largest instance of the problem possible within a fixed time; the amount of work assigned to each processor is held fixed, and the total computational work performed scales up with the number of processors. The figure of merit in this case is called scaled speedup. If each processor's communication is restricted to its immediate neighbors, the ratio of communication time to computation time will be constant: scaled speedup is not limited by Amdahl's law. The Sandia group argued that scaled speedup is the more realistic approach.

It is important to distinguish between scaling up the amount of work and scaling up the size of the problem instance. The algorithms for solving many scientific problems involve work given by superlinear polynomial functions of the

problem size, and thus the problem size can grow only as a sublinear function of the machine's capacity. For example, an algorithm requiring n^2 operations for a problem of size n will need four times as many processors to handle a problem twice as large. Thus parallel speedup produces modest benefits compared to a new algorithms with running times of lower orders. And yet parallel speedup is important because we want to deliver all the computing power available to a problem (*6*).

The Sandia group achieved its results, as we have seen, with an NCUBE/ten, a hypercube consisting of 1024 processors, each with 512K bytes of memory and speed of about 80,000 floating point operations per second. The NCUBE is one of several commercial hypercube machines available, including the Intel iPSC series, the FPS T series, the Ametek Cube, and the Connection Machine. A hypercube consists of $P = 2^n$ processing nodes, named with the n-bit binary numbers $0, \ldots, 2^n - 1$. There is a communication link between two nodes only if their binary numbers differ by exactly one bit, and each node is connected directly to exactly n others. When one processor needs to send a message to another processor, that message must traverse one link for each bit that is different in the processors' binary numbers, being relayed by a series of intermediate nodes. The required protocol is cheap and easy to implement in the interconnection hardware. Each node of a hypercube contains a small control program that is able to send, receive, and relay messages by the protocol.

To map an algorithm onto such a machine so that (ideally) all the processors are fully utilized, it is necessary to divide the problem into parts that can be executed on separate processors and to keep the ratio of communication time to computational time low for each part. (If this can be accomplished by a single method that can be automatically configured for any number P of processors, the algorithm is called scalable.) One of the impediments to this goal on a hypercube is that the time of the longest communication grows as $\log_2 P$; unless the problem itself requires communication only between neighbors, the algorithm may not be scalable.

Many scientific problems involve the solution of partial differential equations over a grid that covers a region of space. These problems are easily mapped to a hypercube by assigning to each node a subregion as large as its memory permits. Because each grid point needs to communicate only with its immediate neighbors in space, and because Gray code mappings can ensure that neighbors on the grid are also neighbors in the hypercube (*7*), the communication time per node is independent of the size of the space. Thus the practical question is how to implement communication so that its time is small.

The Sandia group examined this question for three problems. The first calculates the progress of a two-dimensional acoustic wave through a set of deflectors and provides a graphic display of the resulting heavily-diffracted wavefront. The second calculates dynamic fluid flow in a nonconducting, compressible ideal gas under unstable conditions and displays vortex formations. The

third calculates the deflection of a beam subject to a specified load. The computation for each case consisted of a host program that loaded1024 individual node programs into the machine and their execution. A series of runs of each case for different problem sizes produced measurements of running time and operations completed per second. These measurements showed that the fixed-size speedup on the 1024-processor hypercube was over 500 for each problem (well in excess of the 200 limit in the Karp challenge) and the scaled speedup was over 1000 for each problem.

These impressive results would not have been possible without careful attention to several principles that minimize the communication time experienced by each processor. The assignment of work to each processor was determined during the algorithm's design and remained static during the computation; no dynamic reconfiguration or load-balancing was used. Many exchanges of data between neighboring grid points were carried out simultaneously throughout the machine. Each processor initiated communications with its neighbors in batches that permitted a high degree of overlap; in the wave problem, for example, communications could be completed in about one-third the sum of the individual message-transmission times. On each processor a double-buffering scheme was used to gather data from arrays into a single contiguous buffer prior to transmission of a message, and also to scatter data back into arrays from a contiguous buffer just after reception of a message. Whenever possible, many items of data were lumped into a single message. Finally, the time to load (broadcast) identi-

cal information to all nodes was greatly reduced by a fan-out tree: the first node sends the information to all its nearest neighbors, which in turn relay it to theirs, and so on; every node will have the information after $\log_2 P$ relays. The Sandia group implemented these principles by hand. The challenge now is to find ways of automating them in compilers.

Will these results eventually be extended to all scientific and engineering problems? The answer is clearly no. Problems corresponding to the solution of partial differential equations over a grid are especially suited to the type of partitioning used by the Sandia group. But there are many other types of problems for which load-balanced static partitioning is not as easy. An example is a two-phase image processing problem: in the first phase, the picture is divided into independent chunks for detection of local contours, and in the second, contours are joined across chunks to construct larger features. All the processors can be kept busy during the first phase, but in the second phase, processors whose chunks contain few features will be idle. For computations in which the distribution of computational loads over the processors depends on the input data, dynamic load scheduling is needed to realize the full efficiency of the machine.

The Sandia group has convincingly challenged the assumption hidden in the folklore -- that massively parallel speedup would be elusive in practice. They have given a new impetus to parallel computing.

References

1. A. Karp. 1986. "What price multiplicity?" *Communications ACM 29* (February): 87. (See also comments by J. Rice and C. Bajaj, *Communications ACM 30* (January 1987): 7-9.

2. J. Dongarra, A. Karp, and K. Kennedy. 1988. "Winners achieve speedup of 400." *IEEE Software.* May, pp. 1-5.

3. M. M. Waldrop. 15 Apr 1988. "Hypercube breaks a programming barrier." *Science 240*: 286.

4. G. Amdahl. 1967. "Validity of the single-processor approach to achieving large-scale computer capabilities." *AFIPS Conference Proceedings 30*: 483-485.

5. J. Gustavson, G. Montry, and R. Benner. 1988. "Development of parallel methods for a 1024-processor hypercube." *SIAM J. Scientific and Statistical Computing 9*, 4 (July): 1-32.

6. L. Snyder. 1986. "Type architectures, shared memory. and the corollary of modest potential." *Annual Review of Computer Science 1*: 289-317.

7. P. Denning. 1987. "Multigrids and hypercubes." *American Scientist 75*, 3. May-June, pp. 234-238.

Measuring parallel speedup

Consider solutions to partial differential equations on an $N \times N$ square grid by algorithms that update grid points only from their immediate neighbors. The total computation time required is aN^2, where a is the total time per grid point. Assuming that all communication can be done in parallel throughout the machine (the best case), the total communication time on two or more processors is b, the total time to exchange messages between a pair of neighboring grid points. If this problem is divided equally among P processors, the speedup is

$$F = \frac{\text{Time on 1 processor}}{\text{Time on } P \text{ processors}} = \frac{aN^2}{aN^2/P + b}.$$

F is called fixed-size speedup, because the problem size was held constant while the machine got larger. For many processors, F has a constant asymptote of aN^2/b.

Suppose that the problem size is scaled up to follow the machine size -- that is, $N^2 = cP$, where c is the largest number of grid points assignable to one processor. When one subsitutes the new terms in the formula for F, a new figure of merit results:

$$S = \frac{P}{1 + b/ac}.$$

S is called the scaled speedup, because the total computational work was allowed to scale up with the machine size. Note that S is linear in P.

Suppose that fixed-size speedup were constrained to 200 for practical programs, a limit suggested by Alan Karp's challenge. Since the definition of fixed-size speedup supposes one processor can accommodate the whole problem, $N^2 = c$ for the fixed-size approach and $ac/b = 200$. The scaled speedup for this case is $P/1.005$, which is about half a percent smaller than P.

These formulas are approximate, because the time to load programs and data into the machine has been omitted. In a hypercube, loading time is proportional to $\log_2 P$. Accounting for this term would reduce both figures of merit.

Both the fixed-size and scaled figures of merit can be interpreted as consequences of Amdahl's law that depend on differing assumptions about the relation between problem size and the number of processors.